Here’s what I want to update. The app should be updated so that I can not only ingest a spreadsheet file but connect to a remote SQL database and add that to the list of available sources and have it used to ask a question and get an answer.

I want to be able to connect to one of the top 5 SQL databases (information on those 5 below “Database Connection Guide”). I’d like to see separate files/code for each of these databases.

The process should generally be the same but with some minor differences.

We should use any library or code that is already written, but should not modify the existing code, but create a new set of code specific to remote SQL databases if we can’t fully use an unmodified file/function/library/etc...

Here’s what I’m think the scenario should look like:

The user clicks the “Connect Database” button. Then the user is presented with a list of the top 5 SQL database (information below).

When the user selects on of the 5 SQL databases, the user is asked the relevant questions needed to connect. The connection information (URL, username, password, database, table, etc...) for the remote SQL database should be stored in the main MySQL database so that it can be displayed as available sources alongside the spreadsheets.

Once that information is stored, the remote SQL database is listed in the available sources list, alongside the spreadsheets.

The User can then select that SQL database.

After a question is entered, the SQL database connection information is used to connect to the remote database/table where it gets the schema. Now from here on out we should be able to mostly use our current set of code (except the remote SQL connection) I would like to hear what you have to say… Carrying on, then, like the spreadsheet, the schema and the question is feed to the lllm (that the user has selected) asking it to return the proper SQL query that would be used to answer the question. Then the user is presented with the SQL query and then selects execute query, then the app runs the query against the remote SQL database/table that contains the database information, once the SQL query returns the data, the data and the user’s question is feed to the llm asking it to use that information to provide an answer to the question.

Database Connection Guide

1. MySQL

Installation

npm install mysql2

Connection Pooling

const mysql = require('mysql2');

const pool = mysql.createPool({

host: 'your\_host',

user: 'your\_user',

password: 'your\_password',

database: 'your\_database',

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

module.exports = pool.promise();

Query Execution

async function getUsers() {

try {

const pool = require('./db');

const [rows] = await pool.query('SELECT \* FROM users WHERE status = ?', ['active']);

console.log(rows);

} catch (err) {

console.error('MySQL Query Error:', err);

}

}

getUsers();

Error Handling

pool.getConnection((err, connection) => {

if (err) console.error('MySQL Connection Error:', err);

else {

console.log('Connected to MySQL');

connection.release();

}

});

2. PostgreSQL

Installation

npm install pg

Connection Pooling

const { Pool } = require('pg');

const pool = new Pool({

host: 'your\_host',

user: 'your\_user',

password: 'your\_password',

database: 'your\_database',

port: 5432,

max: 10,

idleTimeoutMillis: 30000

});

module.exports = pool;

Query Execution

async function getUsers() {

try {

const pool = require('./db');

const { rows } = await pool.query('SELECT \* FROM users WHERE status = $1', ['active']);

console.log(rows);

} catch (err) {

console.error('PostgreSQL Query Error:', err);

}

}

getUsers();

Error Handling

pool.connect()

.then(client => {

console.log('Connected to PostgreSQL');

client.release();

})

.catch(err => console.error('PostgreSQL Connection Error:', err));

3. Microsoft SQL Server

Installation

npm install mssql

Connection Pooling

const sql = require('mssql');

const poolPromise = new sql.ConnectionPool({

user: 'your\_user',

password: 'your\_password',

server: 'your\_host',

database: 'your\_database',

options: { encrypt: true, trustServerCertificate: true },

pool: {

max: 10,

min: 2,

idleTimeoutMillis: 30000

}

}).connect();

module.exports = poolPromise;

Query Execution

async function getUsers() {

try {

const pool = await require('./db');

const result = await pool.request().input('status', sql.VarChar, 'active').query('SELECT \* FROM users WHERE status = @status');

console.log(result.recordset);

} catch (err) {

console.error('SQL Server Query Error:', err);

}

}

getUsers();

Error Handling

poolPromise.then(pool => {

console.log('Connected to SQL Server');

}).catch(err => console.error('SQL Server Connection Error:', err));

4. Oracle Database

Installation

npm install oracledb

Connection Pooling

const oracledb = require('oracledb');

async function init() {

try {

await oracledb.createPool({

user: 'your\_user',

password: 'your\_password',

connectString: 'your\_host:1521/your\_service\_name',

poolMax: 10,

poolMin: 2,

poolTimeout: 60

});

console.log('Oracle Connection Pool Initialized');

} catch (err) {

console.error('Oracle Connection Pool Error:', err);

}

}

init();

Query Execution

async function getUsers() {

try {

const connection = await oracledb.getConnection();

const result = await connection.execute('SELECT \* FROM users WHERE status = :status', ['active']);

console.log(result.rows);

await connection.close();

} catch (err) {

console.error('Oracle Query Error:', err);

}

}

getUsers();

Error Handling

oracledb.getConnection()

.then(conn => {

console.log('Connected to Oracle');

return conn.close();

})

.catch(err => console.error('Oracle Connection Error:', err));

5. SQLite

Installation

npm install sqlite3

Connection Pooling

const sqlite3 = require('sqlite3').verbose();

const db = new sqlite3.Database('your\_database.db', (err) => {

if (err) console.error('SQLite Connection Error:', err);

else console.log('Connected to SQLite');

});

module.exports = db;

Query Execution

function getUsers() {

db.all('SELECT \* FROM users WHERE status = ?', ['active'], (err, rows) => {

if (err) console.error('SQLite Query Error:', err);

else console.log(rows);

});

}

getUsers();

Error Handling

db.on('error', err => {

console.error('SQLite Error:', err);

});